TECHNICAL UNIVERSITY OF DENMARK

User Manual

Component Definition Editor

Group 4

05.12.2008

Taha Slime s081738	Agata Balwierz s081172	Steinar Bjornsson s081007
Ali Demirsoy s080653	Kemal Baykal s080654	Tudor Blanaru s083377
Mohamad Gasmi s081195	Martyna Sikorska s081329	Ulyana Tsyukh s080776

Table of Contents

Written and corrected by:

Ali Demirsoy, Taha Slime, Tudor Blanaru, Agata Balwierz, Ulyana Tsyukh

1 Introduction

This user manual provides information on using Component Definition Editor, which is part of the CASE tool project. The aim of the project is to provide developers with the ability to model and simulate embedded systems in a convenient and efficient way using this software. The developer will be able create and visualize embedded systems and test them via simulations before using them in a real systems.

This document describes the correct way of using Component Definition Editor, in order to choose the right tools to construct necessary components correctly. The document contains the overview of the Component Definition Editor and a small tutorial that describes a step-by-step example on creating Component Definition.

2 Overview

The CASE Tool is created by integrating different plug-ins. The Component Definition Editor (Figure 1) is a part of CASE tool system, where a system developer is able to create new components for a system and specify their behavior. It is provided as an Eclipse plug-in and should be loaded together with other plug-ins of the CASE tool

Using Deployment Editor the developer instantiates previously defined components and specifies connections between them. The Dashboard visualizes the behavior of the modeled system. During the simulation one can change the state of different components and the system will react according to the behavior specified during component definition and deployment.

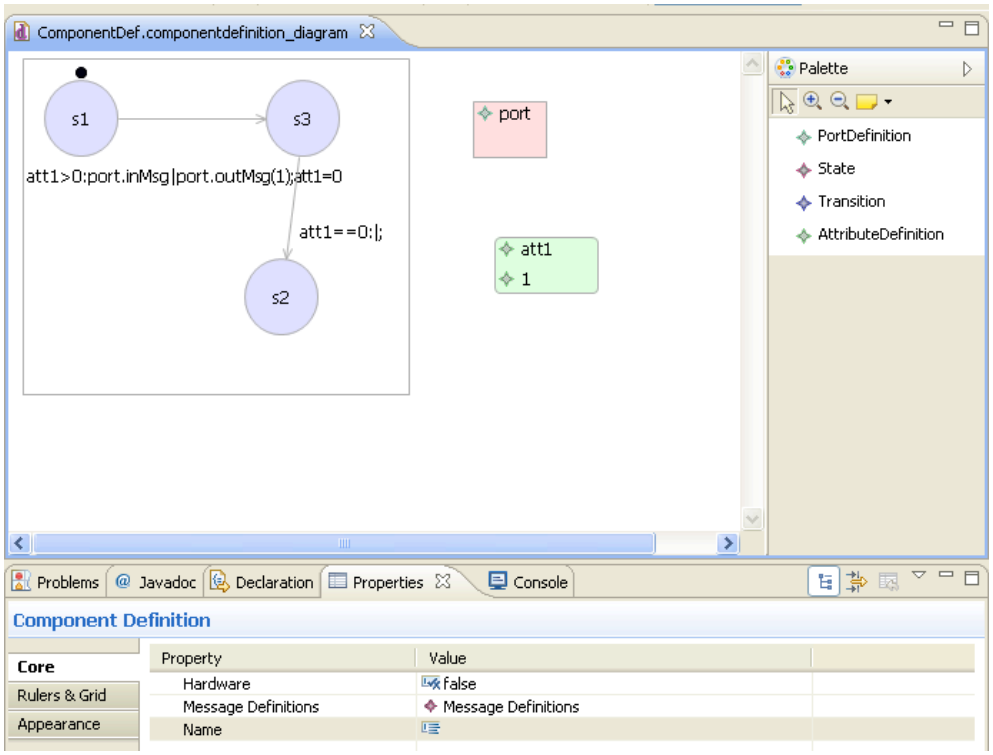

Figure 1: Component Definition editor

3 Step by Step Example

Before we can create Component Definition all plug-ins of the CASE tool should be loaded, for example in the runtime environment.

3.1 New project

First create a new project from the Eclipse runtime environment, select “File” then “New” and then “Project” (Figure 2).

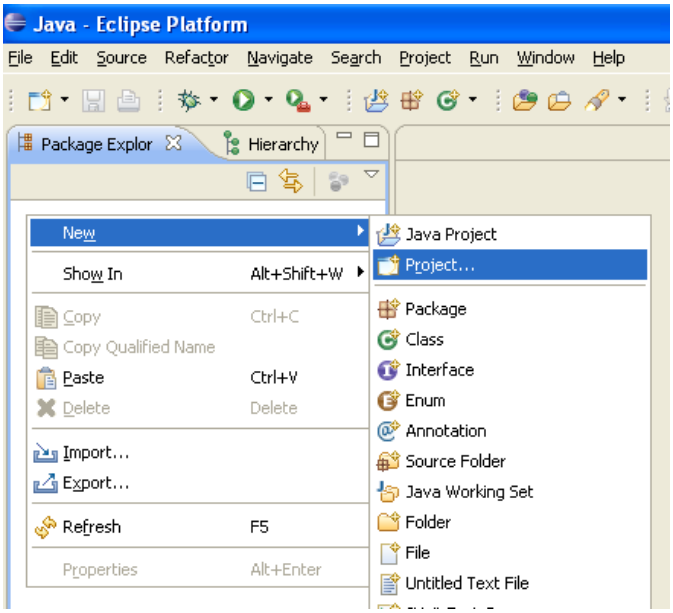

Figure 2: Create empty project

Select the Project wizard (Figure 3) and name project for example “CASETOOL example”.

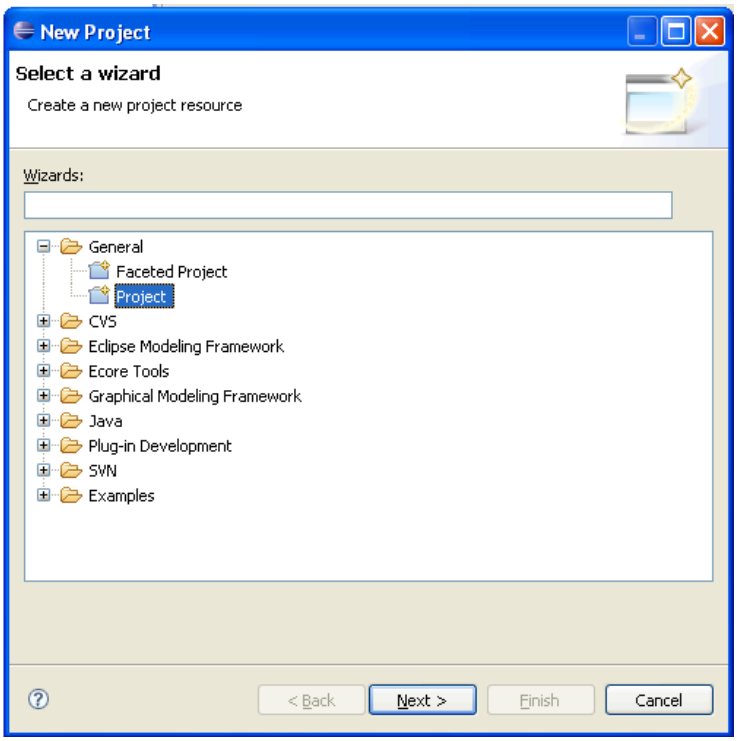

Figure 3: Select Project wizard

At the next figure you can see the project that was created (Figure 4).

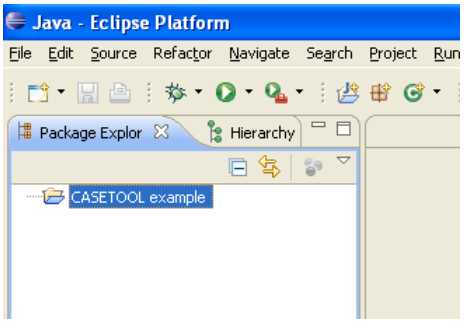

Figure 4: Empty project with the name CASETOOL example

3.2 Component definition diagram

In order to create the Component Definition select `CaseToolComponentDiagram`, which is part of the CASE Tool (SE2 e08) folder (Figure 5). You can name it "`ComponentDef`".

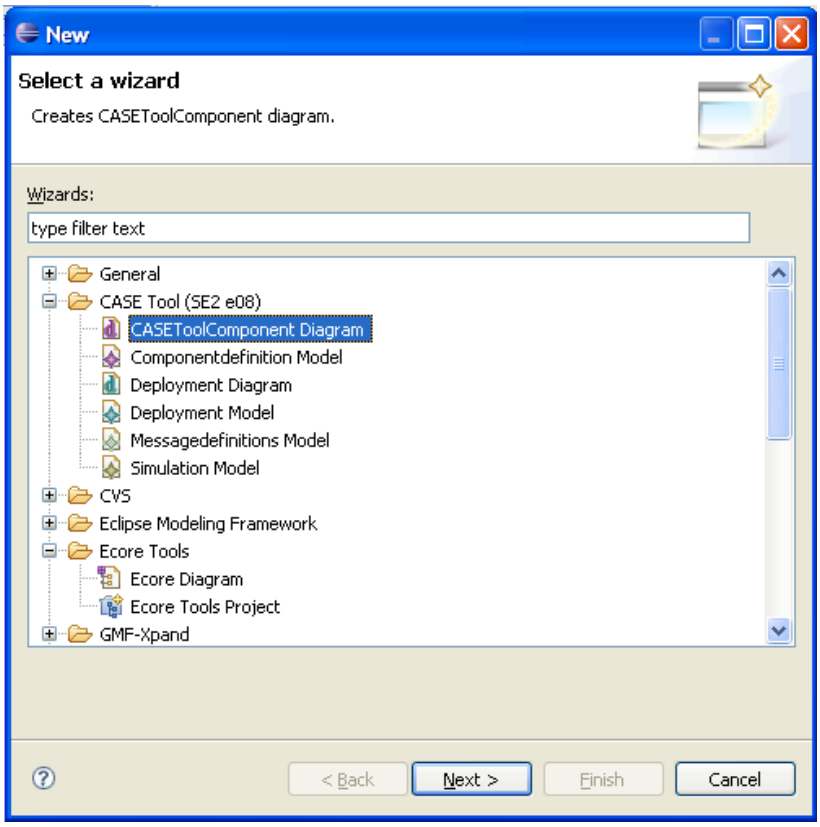

Figure 5: Create new Component Diagram

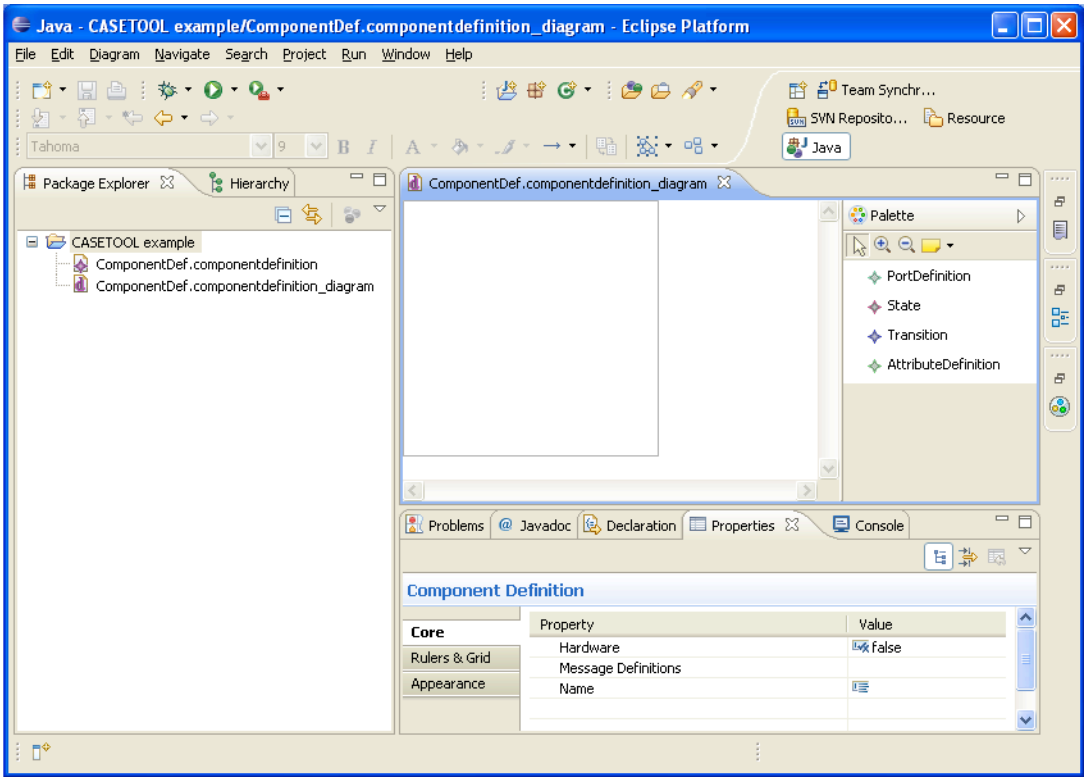

Figure 6: Newly created diagram with an automaton

In the properties tab of this editor you should be able to specify what type of component you want to create. In the following figure the attribute 'Hardware' is set to false which means this is a Software Component. Set it to true if you want to create Hardware component (Figure 7).

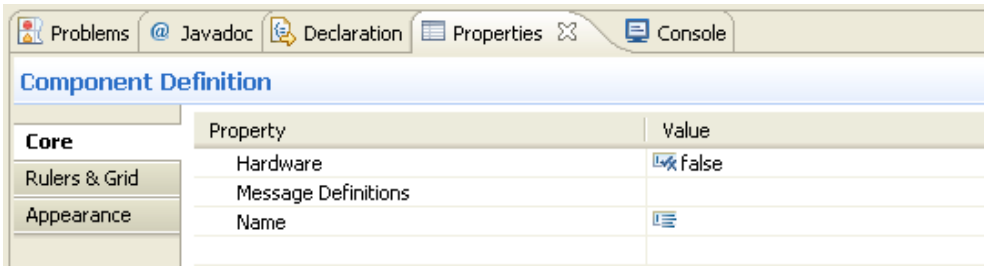

Figure 7: Properties tab of the Editor

The new Component Definition Diagram contains an empty automaton, where you specify component behavior using states and transitions between them.

3.3 Palette

Most of the parts of the Component Definition you can add using “Palette” (Figure 8). It consists of:

- **Port Definition**

  Every component has one or more ports where messages can be send and/or received.

- **Attribute Definition**

  Attribute is a Component variable. It defines in some way the state of the component; it has a name and initial value.

- **State**

  The states describe some “position” that the component is in. The state can be changed by transition. There can be many states, but only one initial state has to be defined in properties tab of the state.

- **Transition**

  Every transition is a connection between two states. It describes what condition can occur and what message should be received so that the component will move to another state. When the component changes the state it can send several messages and make a few assignments to the attributes.

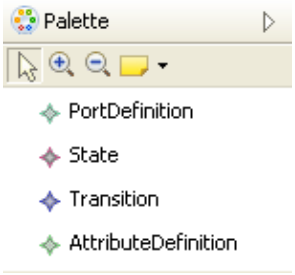

Figure 8: Palette

We will use the Palette to create these components in the next sections.

3.4 State

To add a new state to the diagram drag-drop state from the palette in to the automaton and give them names as in the following figure (Figure 9).

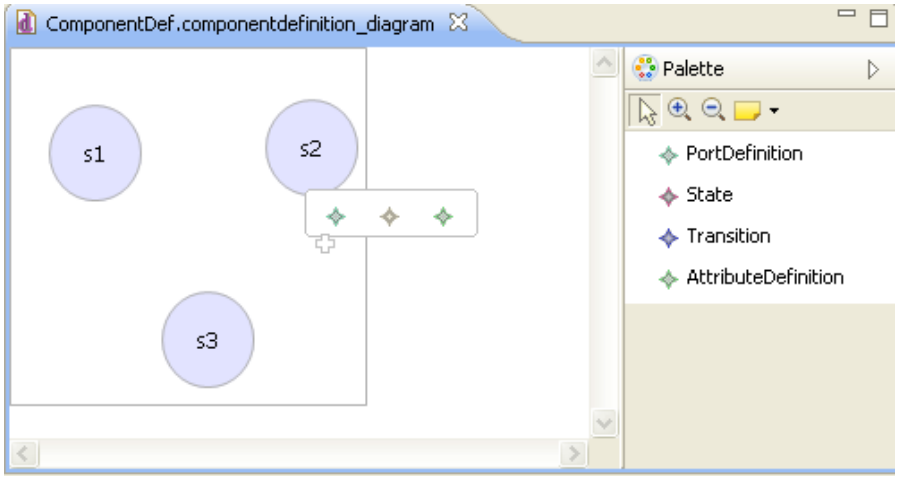

Figure 9: Automaton with its states

To change the name and specify the initial state, click on the state and go to properties tab as in Figure 10. Then you can set initial to true.

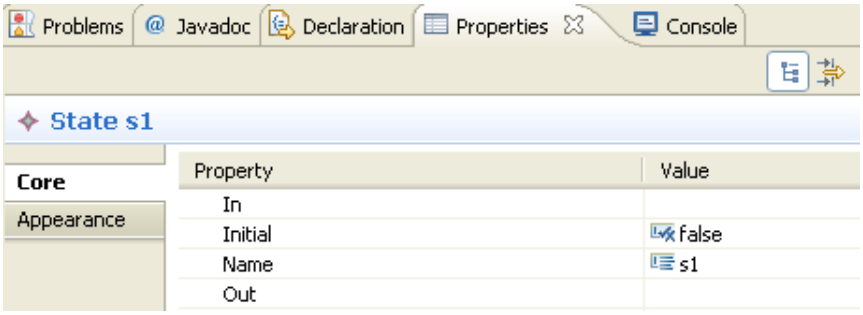

Figure 10: Properties tab of s1 state

After this property is saved the state ‘s1’ is defined as initial and has a special decoration (Figure 11). Every automaton has to have one and only one initial state.

Figure 11: Initial state

3.5 Message definition

To define a message definition, create new “Messagedefinitions” Model (See figure 12) with the name ‘MyMessages’.

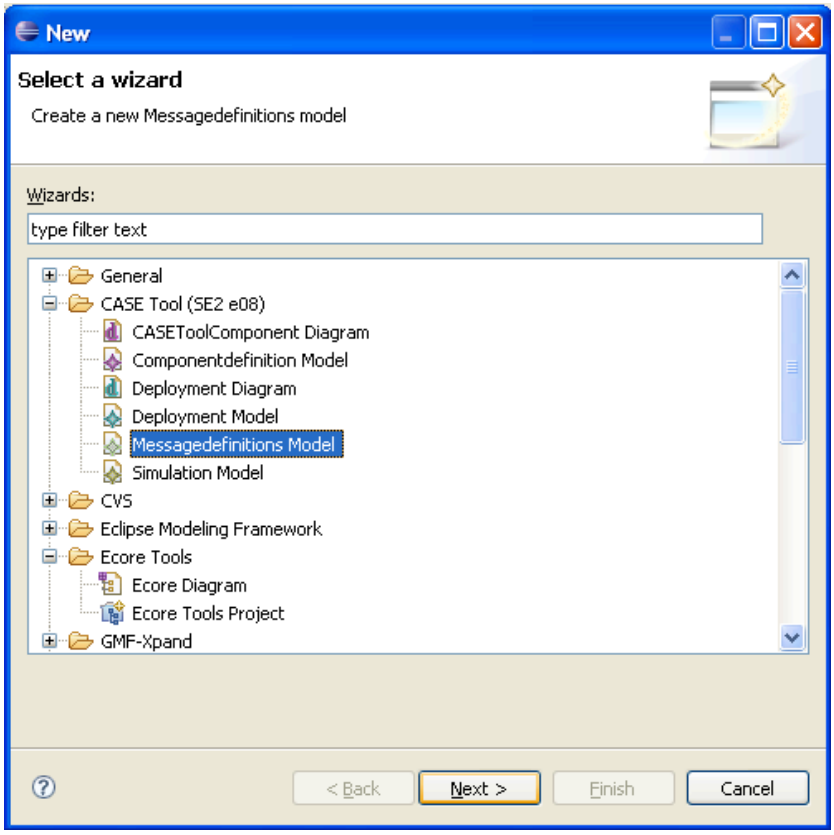

Figure 12: Create Message Definition

In the next step you will be asked to choose the root element. To create more than one message definition, choose “Message Definitions”, otherwise Message Definition (Figure 13).

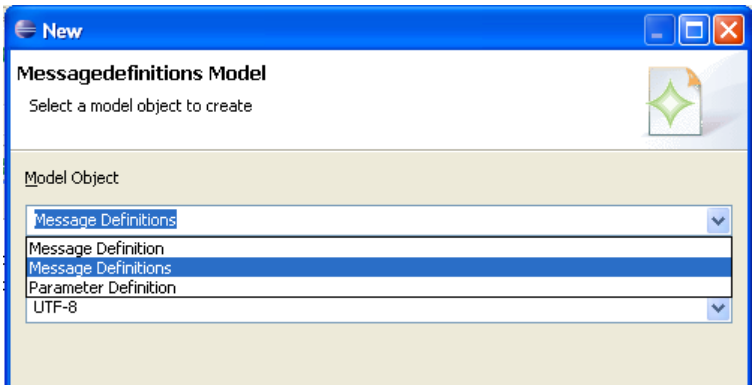

Figure 13: Choose root element

After MessageDefinitions is chosen, to add message definitions simply add a new child to root element and create one or more message definitions (Figure 14).

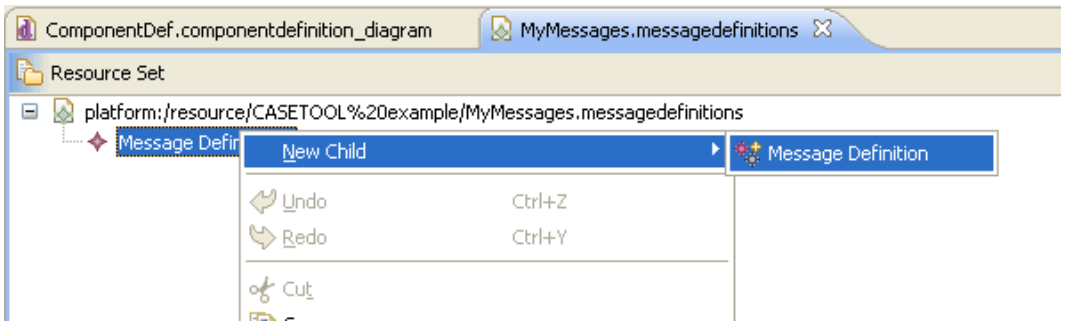

Figure 14: Add Message Definition

In this example we created two message definitions, which are inMsg and outMsg as in Figure 15.

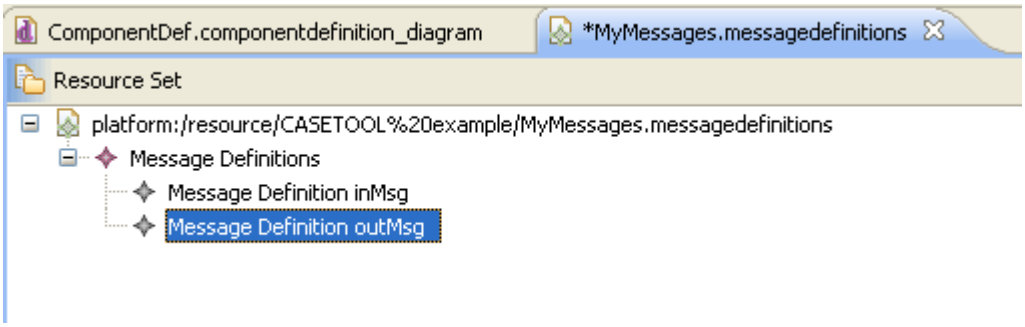

Figure 15: Two message definitions are defined

You can add parameters to each of the definitions by adding new child to these message definitions (Figure 16).

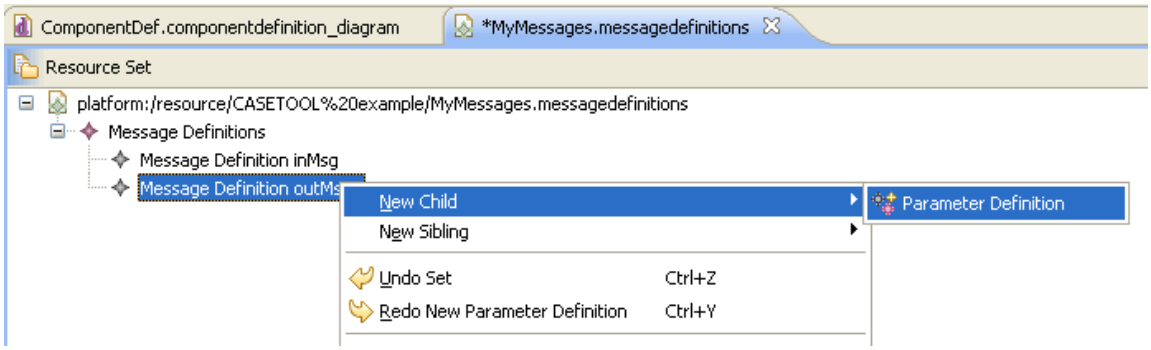

Figure 16: Add parameters to message definitions

To use the created message definitions in your Component Definition diagram, you should go back to the diagram, right click o the canvas, choose “Load Resource” and browse the MessageDefinition file (Figure 17).

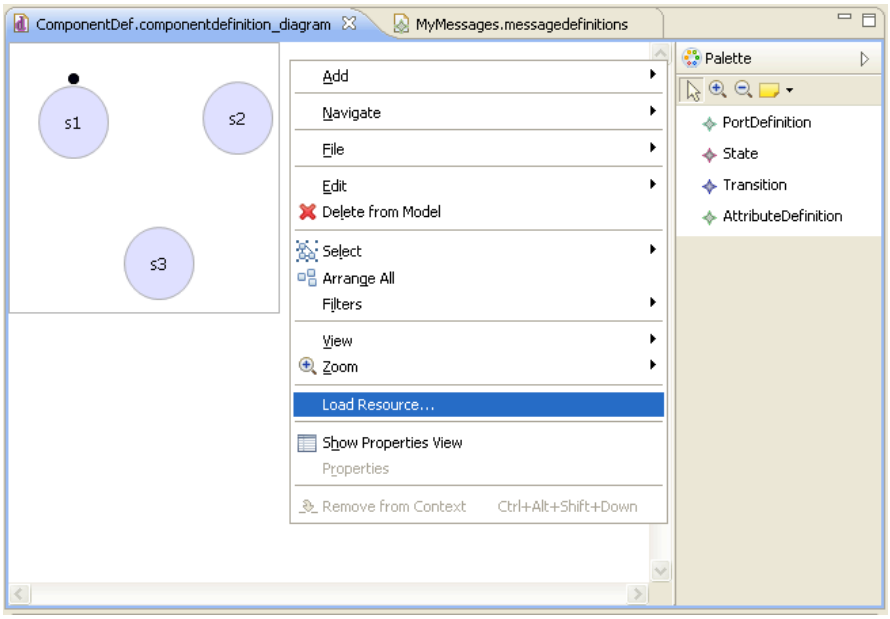

Figure 17: Load MessageDefiniton files

3.6 Port definition

Now we can create the port and attribute definitions. Here we created a port definition named “port” and attribute named attr1 (See figure 18).

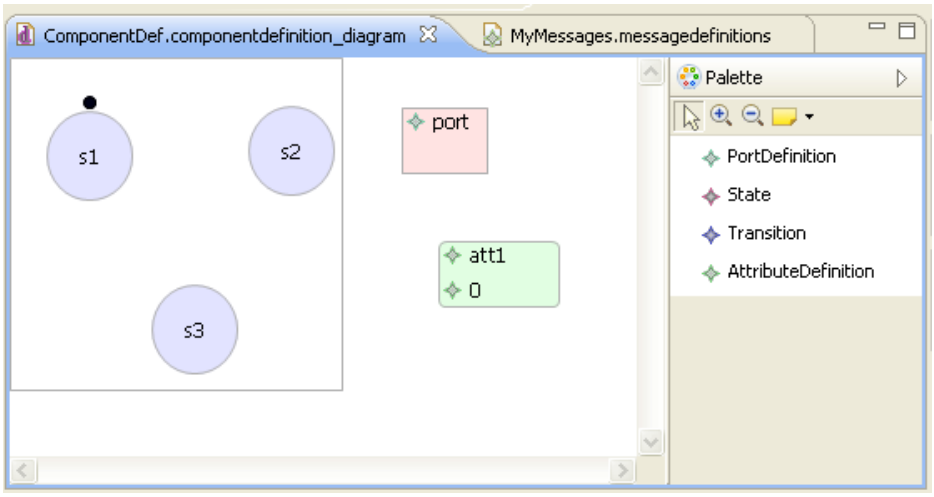

Figure 18: Automaton, Port and Attribute definitions

We will use these message definitions by defining the ports in and out messages. You can configure the port definition by opening its properties tab (Figure 19), where you can add in and out message types.

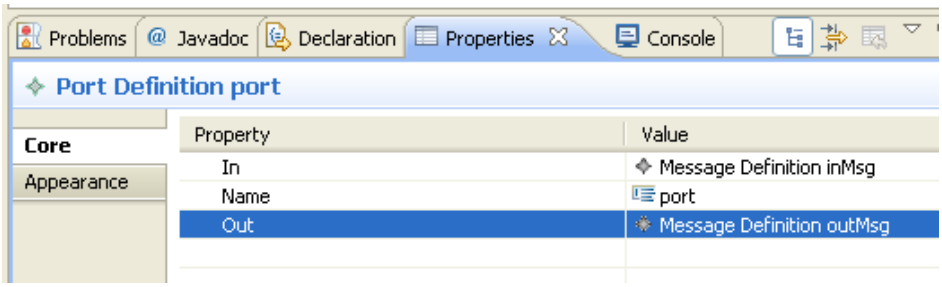

Figure 19: Specify message definitions of port

3.7 Transition

Now we are able to add our transitions in between attributes. To create the transitions, drag and drop the transition node between two states. When you drop the node, you will be asked to fill in the label. Transition consists of one condition, one incoming message, zero, one or many outgoing messages and assignments.

Condition:**InMessage**|**OutMessages**;**Assignments**

A valid example of a transition can be seen in Figure 20:

Figure 20: A valid example of transition label

Where: `attr` – is the attribute that we just specified
`P1` – is parameter from `port.inMsg`

In this section the detailed specification of the Condition, inMessage, outMessage and Assignments is given. Note if the label you have entered is incorrect you will see the error message.

3.7.1 Condition

Condition is a Boolean expression, where the following notations can be used:

1. Operators:

Table 1: Available operators

Name of operator	Symbol
Logical *and*	&
Logical *or*	|
Logical *nand*	!&
Logical *nor*	!|
Logical *xor*	^
Logical *not*	!
Relational *equal to*	==
Relational *not equal to*	!=
Relational *greater than or equal to*	>=
Relational *less than or equal to*	<=
Relational *greater than*	>
Relational *less than*	<
Arithmetic *addition*	+
Arithmetic *subtraction*	-
Arithmetic *multiplication*	*
Arithmetic *division*	/

2. Integer constant
3. Brackets: "(" and ")"
4. Attributes (first attribute has to be defined in component definition, if it has a name "att1" in the label you will use string "att1")
5. Parameters (the names of parameters of the incoming message)

6. Arithmetical expressions

3.7.2 InMessage

In this part you can specify the port and which message can be received.

Example:

```
port1.message1
```

3.7.3 OutMessage

In this part you specify which messages you can be sent. There can be zero, one or many Out Messages. They should be separated by coma (",").

One OutMessage consists of NameOfPort.NameOfMessage(parameter1, parameter2)

NameOfPort – port where the message has to be sent

NameOfMessage – the message

parameter1, parameter2 – parameters for the message, they are optional and are represented as arithmetic expressions that can contain the following notations:

1. Arithmetic operators (Table 1)
2. Attributes
3. Parameters of in message (look at condition)
4. Brackets "(", ")"

Example of that parameter: `par1+12/34*6`

Example of outMessage:

```
port1.inMess1(par1+6,1),port1.inMess2(a),port3.inMess3
```

Note: there have to be exactly the same number of parameters as in the message description.

3.7.4 Assignments

There can be zero, one or more assignments. Assignments are separated by coma "**,**" and look like Assignment1**,** Assignment2.

Assignment consists of NameOfAttribute=arithmeticalExpression:

NameOfAttribute – attribute that you want to assign value to

arithmeticalExpression – the specification is the same as in OutMessage.

3.8 Saving diagram

After all the steps you will be completed your component diagram you can safely save and exit. The last appearance of the Component Definition will be as in the next figure:

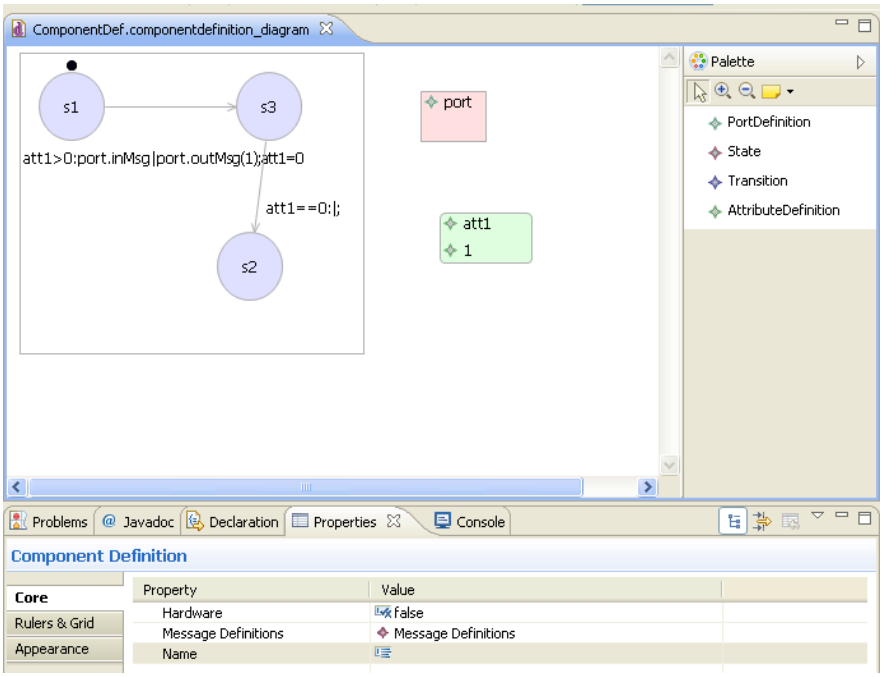

Figure 21: Complete Component Definition diagram

The specified Component Definition diagram can be used in the Deployment, which is the next step in the developing of the CASE Tool system.

4 Simulation algorithm

The simulation algorithm provided as a plug-in can be used by the Simulation Manager to simulate a deployment.

4.1 High level overview of the steps

The simulation algorithm consists of the following steps:

- **Parse the deployment and create all run-time instances –** first discover the components and ports, then busses and in the end the connections. All this is done when a new instance of the simulation is created.
- **Simulate components** – this is the first sub-step of one simulation step.
- **Simulate busses** – this is the 2nd sub-step of one simulation step.

4.2 Components behavior

This paragraph describes how each type of elements definition in a deployment – like component definition, port and message definition, parameter and attribute definition – and

sub elements of a component definition – like automaton, state and transition – behaves and/or affects the simulation algorithm. We will use the term "*element is simulated*" to refer to what happens with/for the particular element when one step of the simulation is performed.

4.2.1 Component (definition)

A component is simulated by selecting the current state of the automaton and performing the first transition that apply – that is, the first transition for which the condition is evaluated to true and/or the required incoming message exist on the required port.

4.2.2 Automaton

The automaton is used by the simulation to determine the initial state and set it as current – when the run-time component instances are created from the deployment – and for getting the current state when simulating one step for a component.

4.2.3 Transition

A transition is selected (applies) when the incoming message specified in the transition label is found in the specified port of the component and when the condition specified in the label is satisfied. The result of performing the transition can be either, one, or both of the following to actions: messages are sent through the component's ports and assignments are performed for component's attributes.

In case multiple transitions could be selected, the algorithm will select the first one in the order in which the transitions were defined in the automaton.

4.2.4 Ports

Ports are used to hold in a buffer the incoming messages and for sending – by providing a connection to the simulation algorithm to be used for finding the destination port - the messages that the simulation creates as a result of a transition being performed.

A message is removed – consumed – from a port only in case a transition that had that message as the trigger was performed. This could lead to a buffer overrun condition of a port in case the definition of the automaton for the component is not properly specified.

4.2.5 Connections

Connections are used by the simulation to determine inside which port's buffer a message should be transferred. In case the connection is via a bus, the actual delivery of the message will take 2 simulation steps because the message will first be moved into the buffer and then to the destination port.

Connections are not oriented connection – meaning that the simulation will send a message for a port for both the in and the out connections.

4.2.6 Attributes

Attribute definition default value is used for initializing the value of the attribute for the simulation.

Attribute names inside the same component definition should be unique because of the transition label parsing. This is not validated and having duplicated attributes names may result in unexpected behavior of the simulation.

4.2.7 Messages

Message definitions contain the name of the message and a list of parameter definitions. During the simulation, these will be used to create run-time instances of a message. These message run-time instances are sent through the ports between components of the simulation. The value for each of the parameters of a message is evaluated based on the context of the message – context meaning the component run-time instance where the message exists at the time of evaluation.

Message definitions should have unique names, especially when used in the same ports.

4.3 Running a simulation

The steps described below are to be followed when running a simulation for a deployment:

- Open the deployment diagram, right click on it and select *Start Simulation By Group 2*

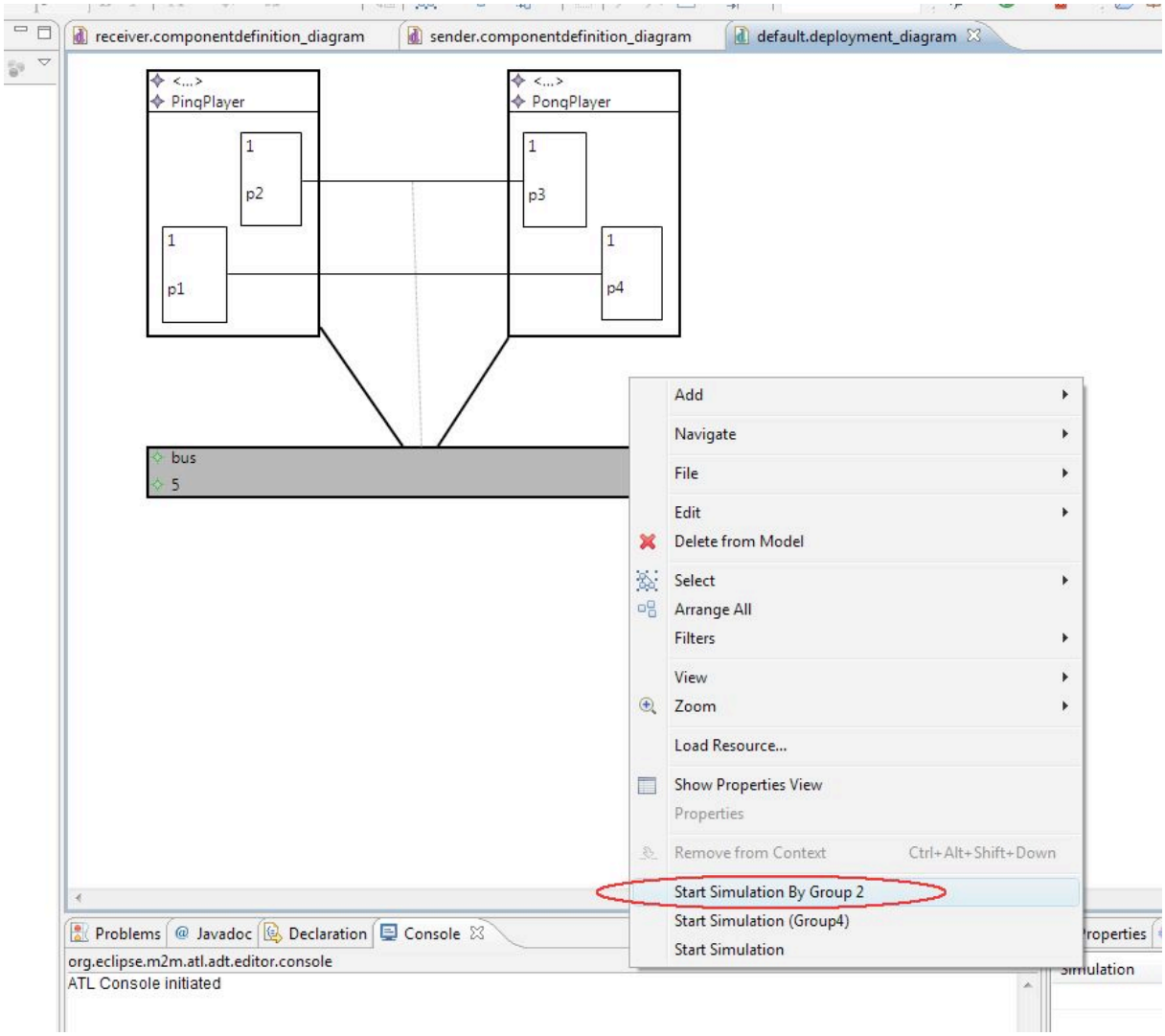

Figure 22: Start Simulation by Group2

- In the *Get Simulation Dialog* provide a name for the simulation instance and select an algorithm from the drop-down list:

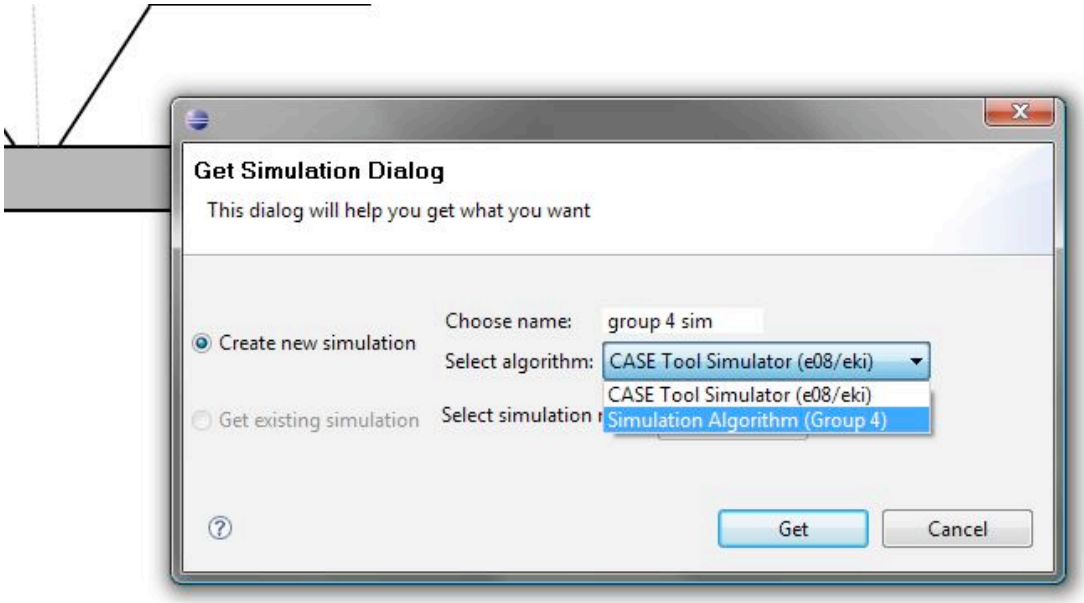

Figure 23: Select algorithm for the simulation

- Interact with the simulation by clicking on the arrow icons next to the simulation instance and observe the states of the simulation's objects in the tree-view:

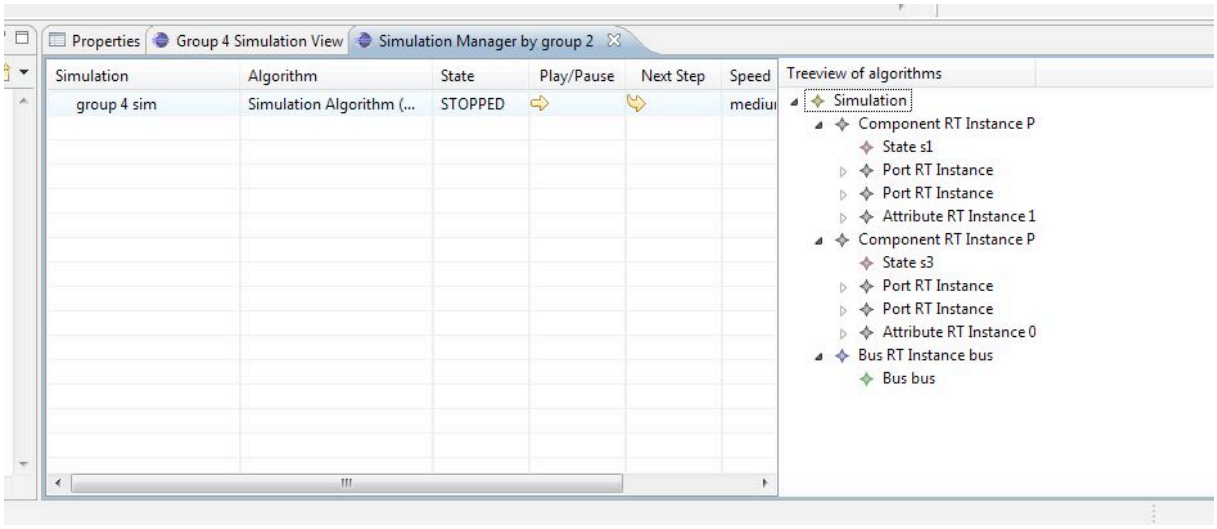

Figure 24: Simulation results